VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

NAVIGACE V GRAFU

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE Vojtěch Žák
AUTHOR

BRNO 2009

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

NAVIGACE V GRAFU

NAVIGATION IN GRAPH

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — Vojtěch Žák
AUTHOR

VEDOUCÍ PRÁCE — Ing. Čermák Martin
SUPERVISOR

BRNO 2009

Abstrakt

Reprezentace problémů nebo systémů je důležitá. Čím unifikovanější reprezentaci získáme, tím snáz poté nalezené řešení či operace zaznamenáme a předáme. Jednou z nejpoužívanějších reprezentací systémů je graf. Pro graf jsou definovány jasné pravidla a pro graf jsou také definovány některé algoritmy. Tato práce se bude zabývat právě skupinou takovýchto algoritmů. Konkrétně algoritmy pro prohledávání stavových prostorů. Pomocí poznatků z těchto algoritmů se sestaví nový algoritmus nad grafem, reprezentujícím areál fakulty, pro vyhledávání nejkratších cest. Tento algoritmus poté uplatní ve výsledné aplikaci.

Abstract

The representation of problems and systems is important. The more unificated representation we get, the more precise operations and solves we can write down and preserve. One of the mostly used representations of systems is graph. For graph there are defined ground rules as well as ground algorithms. This work is oriented on part of these algorithms. Specifically, algorithms for searching the state space. With the knowledge from these algorithms a new algorithm is assembled. An algorithm for finding the shortest paths over graph, representing the area of this faculty. The algorithm then applies the resulting application.

Klíčová slova

Graf, vyhledávací metoda, optimální funkce, faktor couvání, podmínka expanze, bridge uzel.

Keywords

Graph, looking up method, optimal function, the reverse factor, expand condition, bridge node.

Citace

Žák Vojtěch: Navigace v grafu, bakalářská práce, Brno, FIT VUT v Brně, 2009

Navigace v grafu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Čermáka Martina. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Vojtěch Žák
3.5.2009

Poděkování

Děkuji mému vedoucímu práce Ing. Čermákovi Martinovi stejně jako mému konzultantovi Ing. Koutnému Jiří za pomoc a vedení při vypracování bakalářské práce.

Obsah

Úvod

Tato práce vznikla za podpory fakulty informačních technologií vysokého učení technického v Brně. Je zaměřena na odvětví umělé inteligence, konkrétně na navigaci v grafech. V této práci se pokusím co nejlépe popsat přípravu, vývoj a aplikaci nového algoritmu pro vyhledávání nejkratší cesty v grafu. Jako reálný objekt pro reprezentaci grafu, nad kterým bude tento algoritmus fungovat, byl vybrán areál fakulty informačních technologií.

Co jsou grafy?

Ještě před matematickou definicí grafu je vhodné si říct, co grafy jsou a k čemu se využívají. Grafů rozeznáváme celou řadu typů, v této práci však nepůjde ani o grafy využívané ve statistice (sloupcový, koláčový…), ani o grafy funkcí.

Grafy si můžeme představit jako zjednodušení reálného světa, kde studovaný problém znázorníme pomocí bodů a čar, které je spojují, a tím popisují vlastnosti. Takovým bodům pak v teorii grafů říkáme vrcholy grafu a čáry, které je spojují, nazýváme hrany grafu.

Obsah práce

Kapitola první se zaměřuje na úvod do problematiky grafů. Graf je struktura pro tuto práci klíčová. Po uvedení do této problematiky je definován objekt, na kterém se budou možnosti tohoto systému prezentovat. Dále je přesně definován graf reprezentující tento reálný objekt.

Na problematiku grafů navazuje plynule úvod do metod řešení problémů v kapitole druhé. Tato kapitola se dále zabývá samotným jádrem této práce, a to sice postupným vývojem konečného algoritmu z poznatků o grafech a známých vyhledávacích metodách.

Uvedení tohoto konečného algoritmu do praktického prostředí výsledné aplikace a také popis a stručná dokumentace této aplikace je k naleznutí v kapitole třetí.

Na závěr této práce se pokusím objektivně zhodnotit dosažené výsledky a v neposlední řadě se zaměřit na případný budoucí vývoj této aplikace.

1 Teorie grafů

Historie teorie grafů

Za zakladatele teorie grafů je považován Leonhard Euler (1707-1783), který roku 1736 publikoval řešení příkladu *Sedmi mostů města Königsbergu* (Královce). Grafy pak zůstávaly přes sto let na okraji zájmu matematiků - vrátili se k nim až roku 1847 Gustav Kirchhoff (1824-1887), když se zabýval výpočtem proudů v elektrických sítích pomocí počtu koster grafu, a 1857 Arthur Cayley (1821-1895), který pomocí grafů zjišťoval počty různých uskupení molekul alkanů. Nejznámější úlohou teorie grafů v 19. století byl *problém čtyř barev* - otázka zněla, zda-li je možné každou mapu obarvit pomocí čtyř barev tak, aby sousední státy měly různé barvy, který byl kompletně vyřešen až roku 1976. Největší vývoj v oblasti teorii grafů se však projevil ve 20tém století. Významných poznatků bylo dosaženo i v Československu. V roce 1926 publikoval Otakar Borůvka (1899-1995) svůj algoritmus

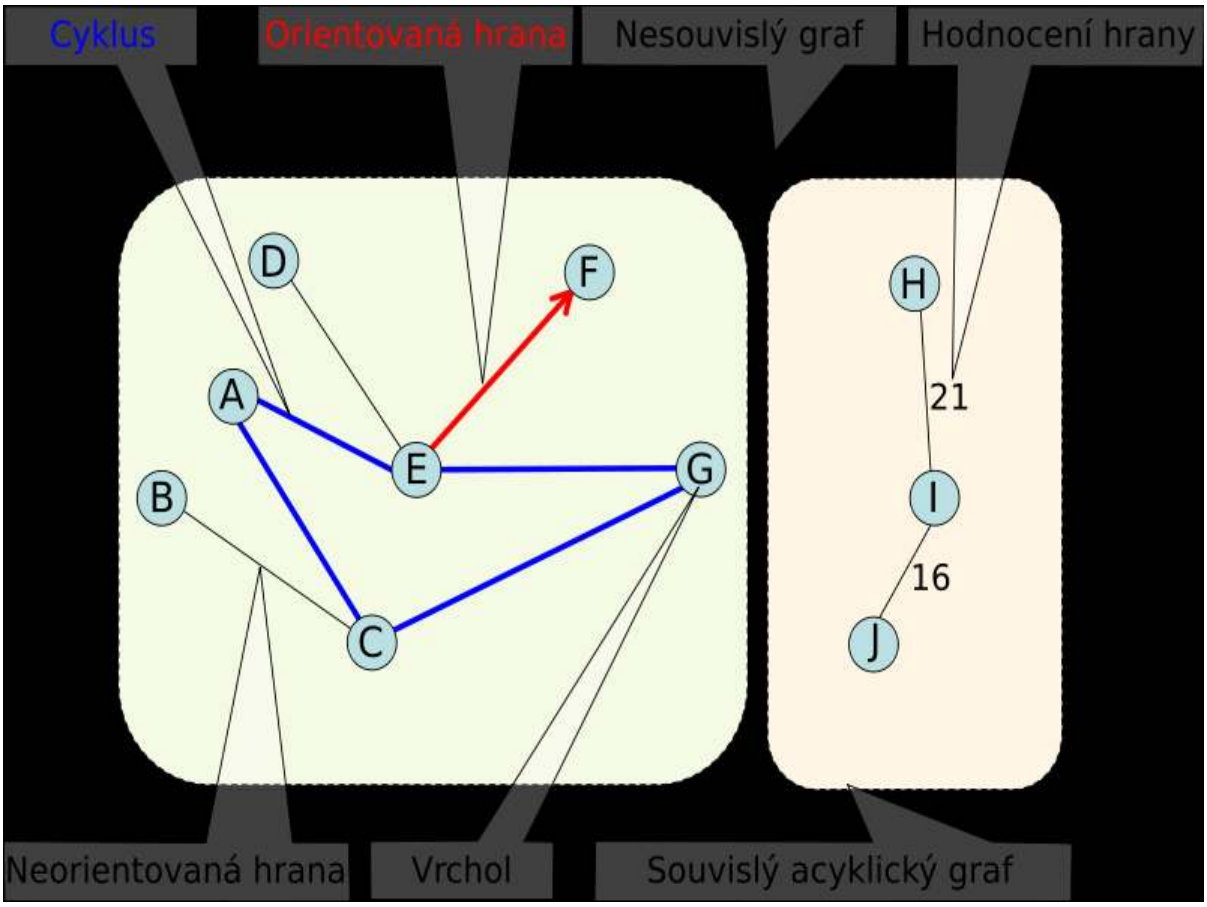
pro nalezení minimální kostry. Tento problém měl praktický důvod - co nejvýhodnější výstavbu elektrických sítí. Stejný problém vyřešil (jiným algoritmem) v roce 1930 také Vojtěch Jarník (1897-1970).

1.1 Obecný graf

Graf je základním objektem teorie grafů. Graf je struktura tvořená dvojicemi *(V, E)*, kde *V* je nějaká neprázdná množina vrcholů a *E* množina některých dvojic prvků z *V*.

- Prvky množiny *V* se nazývají vrcholy (nebo také uzly) grafu.
- Prvky množiny *E* se nazývají hrany grafu.

Tyto hrany grafu spojují vždy dva vrcholy (uzly) grafu. Jelikož je graf struktura nejlépe vystižitelná graficky, následuje obrázek obecného grafu vysvětlující nejzákladnější pojmy.



Obr. 1 Obecný graf

1.1.1 Vlastnosti grafů

Graf jako definovaná struktura respektuje určitá pravidla. Tyto můžeme zapsat jako vlastnosti grafu. Mezi základní vlastnosti grafů patří:

- *prvek i sousedí s prvkem j*, pokud z *i* vede hrana do *j*
- relace ρ z množiny E do V se nazývá *incidenční relace*; je-li hrana *h* v relaci s vrcholem *u*, *u* je *incidentní* s *h*.
- je-li $|\rho(h)| = 1$, hrana h je *smyčka*.
- platí-li, že $\rho(h_1) = \rho(h_2)$, h_1 a h_2 jsou *rovnoběžné hrany*

- *Sled v grafu* je posloupnost vrcholů a hran $v_0\ e_1\ v_1\ e_2 \ldots e_n\ v_n$,kde hrana e_i má koncové vrcholy v_i a v_i

- *Cesta v grafu* je sled, ve kterém se neopakují vrcholy.

- *Souvislý graf* je takový, že mezi každými dvěma vrcholy existuje sled (cesta).
- Graf *H* se nazývá *podgraf* grafu *G*, jestliže V_H je podmnožinou V_G a E_H je podmnožinou E_G.
- *Komponenta* grafu je jeho maximální souvislý podgraf.
- Pro libovolný graf existuje *cyklomatické číslo grafu*, pro které platí:

  $$C(G) = |E| - |V| + p$$

  kde *E* je množina všech hran, *V* je množina všech vrcholů a *p* je počet komponent grafu. Cyklomatické číslo C(G) značí počet nezávislých kružnic v grafu. Je-li C(G) = 0, pak graf neobsahuje kružnice.

Obecně pro tvorbu algoritmu pro vyhledávání v grafu platí, pokud je cyklomatické číslo rovno nule, tedy jedná se o necyklický graf, řešení algoritmu bude nejjednodušší. Pokud, jako v tomto případě, cyklomatické číslo rovno nule není, platí čím menší toto číslo, tím efektivnější algoritmus. Příkladem grafů s cyklomatickým číslem rovným nule jsou stromy. Stromy jsou specifické grafy, které jsou při vhodném rozvržení svých uzlů vhodné pro třízení informací (např. binární stromy).

1.1.2 Rozdělení grafů

Na základě různých kritérií grafu můžeme grafy rozdělit do více skupin:

Podle orientace hran

- orientované (hrany jsou uspořádané dvojice vrcholů)
- neorientované (hrany jsou dvouprvkové množiny vrcholů, tedy

Podle četnosti hran

- Prosté grafy - tyto grafy nemají rovnoběžné hrany, tzn. mezi libovolnými dvěma vrcholy vede nejvýše jedna hrana
- Multigrafy - mezi dvěma vrcholy multigrafu může vést libovolný počet hran; graf je pak definován jako $G = (V,E,\varepsilon)$. Hrana je abstraktní objekt a funkce ϵ určuje, které dva vrcholy tato hrana spojuje

$$\epsilon : E \to \binom{V}{2}$$

- řídké grafy - grafy s relativně malým množstvím hran (vzhledem k počtu vrcholů).
- husté grafy

Podle existence kružnice v grafu

- cyklické
- acyklické (např. stromy)

Podle souvislosti

- souvislé (existuje-li cesta mezi každou dvojicí vrcholů)
- nesouvislé

Podle toho, zda lze graf nakreslit do roviny bez křížení hran

- rovinné (též planární)
- nerovinné

Další důležité rozdělení

- neohodnocené grafy
- ohodnocené grafy, kde každé hraně přísluší nějaká další informace — typicky číslo, u grafů popisujících stavový automat hodnota přečtená ze vstupu. Obecně mluvíme o „ohodnocovací funkci“ - zobrazení, které každé hraně z množiny hran přiřadí „hodnotu hrany“. Hodnota

hrany může např. znamenat vzdálenost mezi vrcholy, kapacitu hrany pro přenos (informací, komodit) a jiné.

1.1.3 Reprezentace grafu

Graf, pokud se jedná o rovinný, je nejlépe vystihnutelný graficky, tedy nakreslením. Pokud se ovšem nejedná o rovinný graf, nebo se jedná o velmi rozsáhlý a složitý multigraf, je možné a vhodné využít alternativní reprezentace grafu.

Možné **reprezentace grafu** tedy jsou**:**

- *„obrázkem“*, správně řečeno *nakreslením*
- *maticí sousednosti*: je-li |*V*| = n, pak čtvercová matice sousednosti *A* je typu $\{0,1\}^{n\times n}$ a platí $A_{i,j} = 1 \Leftrightarrow \{i,j\} \in E$
- *maticí vzdálenosti*: jsou-li hrany grafu ohodnocené, lze výše uvedenou matici modifikovat tak, že místo jedničky je na daném pozici uvedeno ohodnocení (délka) příslušné hrany
- *Laplaceovou maticí*: opět čtvercová matice, tentokrát typu $\mathbb{Z}^{n\times n}$, pro niž platí

$$(L_G)_{i,j} = \begin{cases} deg(i), & i = j \\ -1, & \{i,j\} \in E \\ 0, & i \neq j \ \wedge \{i,j\} \notin E \end{cases}$$

deg(i) je počet hran, které začínají nebo končí ve vrcholu *i*, tzv. stupeň vrcholu

- *maticí incidence*: je-li |*V*| = n a |*E*| = m, pak matice incidence je typu $\{-1,0,+1\}^{n\times m}$ a platí

$$(I_G)_{i,e} = \begin{cases} -1, & e = (i,\bullet) \\ +1, & e = (\bullet,i) \\ 0, & \text{jinak} \end{cases}$$

Jinými slovy, každá hrana má -1 u vrcholu, kde začíná a +1 tam, kde končí. U neorientovaných grafů je na obou místech +1.

- *seznamem sousedů*: je-li |*V*| = n, uspořádáme vrcholy grafu do pole velikosti n a v i-tém prvku tohoto pole bude ukazatel na spojový seznam vrcholů, které s vrcholem *i* sousedí. Toto uspořádání je vhodné při množství údajů menším než n^2. Tj. při ostře menším počtu hran než n^2, kde n je počet vrcholů. Jedná se o tzv. řídké grafy, kterých je v prostředí sítí nejvíce.

1.1.4 Identifikace použitého grafu

Jedním z bodů zadání této bakalářské práce je vybrat reálný objekt pro reprezentaci navigace v grafu. Jako takovýto objekt byl vybrán areál naší fakulty (Fakulta Informačních Technologií), ve kterém výsledná aplikace vyhledává nejkratší cesty mezi dvěma body (uzly grafu).
Výsledným grafem pro reprezentaci budov naší fakulty bude tedy graf:

- ohodnocený
- cyklický
- neorientovaný
- nerovinný
- souvislý
- multigraf

kde uzly grafu budou reprezentovat jednotlivé místnosti, vchody a křižovatky. Hrany grafu poté zpravidla chodby, popř. nejkratší vzdálenosti přes otevřená prostranství. Graf bude stylizován tak, aby mezi dvěma sousedními uzly neexistovala křivka, ale vždy jen přímka.

Vícepodlažní budova se ovšem dá reprezentovat pouze grafem nerovinným. Bylo by tedy vhodné pro její reprezentaci použít jednu z výše uvedených reprezentací.

Avšak pro nejlepší (nejčastější) grafickou reprezentaci navigace v určité budově (a navigace všeobecně) je vhodné použít právě rovinný graf. Pro řešení tohoto problému je využito předpokladu, že výšková vzdálenost mezi dvěma podlažími v rámci jednoho areálu je konstantní. Jinými slovy, že je nutné urazit stejnou vzdálenost pro překonání výškového rozdílu dvou podlaží. Různá podlaží mohou mít různou výšku. Díky tomuto předpokladu budeme schodiště mezi podlažími reprezentovat jako uzel grafu. Reprezentaci nerovinného grafu tedy rozdělím na vícero reprezentací grafu rovinného právě v rámcích těchto podlaží.

Proto výsledná aplikace bude vykreslovat vždy jen jedno podlaží. Cesta zasahující do více podlaží bude vykreslena na více obrazovkách.

Více v kapitole 4.

2 Vyhledávání v grafu

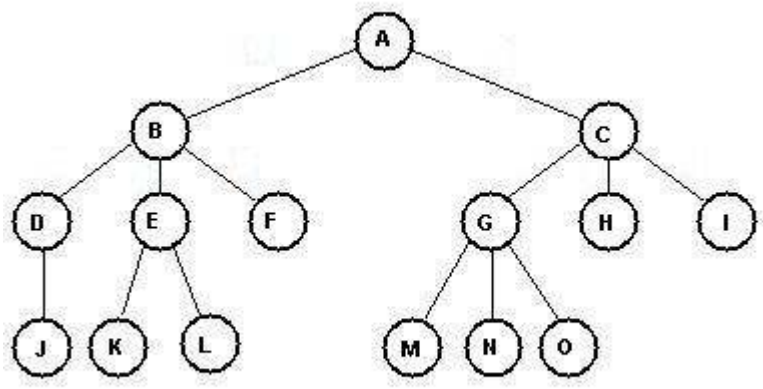
2.1 Neinformované metody

Neinformované metody nemají k dispozici žádnou informaci o cílovém stavu a nemají ani žádné prostředky, jak aktuální stavy hodnotit. Podobné metody musí někdy používat i člověk například při prohledávání mapy, hledá-li cestu z nějakého (výchozího) místa do jiného (cílového) místa a nemá-li vůbec žádnou představu, ve kterém směru od výchozího místa cílové místo leží.

více informací naleznete v [1].

Neinformované metody mají své silné stránky v jednoduchosti jejich algoritmů a širokou škálou použití. Jejich nevýhodou je ovšem náročnost vyhledávání při rozsáhlejších grafech. Nicméně algoritmy těchto metod tvoří základ pro metody následující.
Mezi základní neinformované metody, a mezi základní vyhledávací algoritmy vůbec, patří metody BFS (Breadth First Search) a DFS (Deap First Search).

Funkčnost těchto dvou základních algoritmů budu demonstrovat na následujícím stromovém grafu. Počáteční uzel grafu je uzel „A“. Hledaným výsledným uzlem je uzel „I“.



Obr. 2 Stromový graf

2.1.1 Metoda slepého prohledávání do šířky (BFS)

Základní algoritmus metody BFS je následující:

1. Sestroj frontu OPEN (bude obsahovat všechny uzly určené k expanzi) a umísti do ní počáteční uzel.

2. Je-li fronta OPEN prázdná, pak úloha nemá řešení a ukonči proto prohledávání jako neúspěšné. Jinak pokračuj.

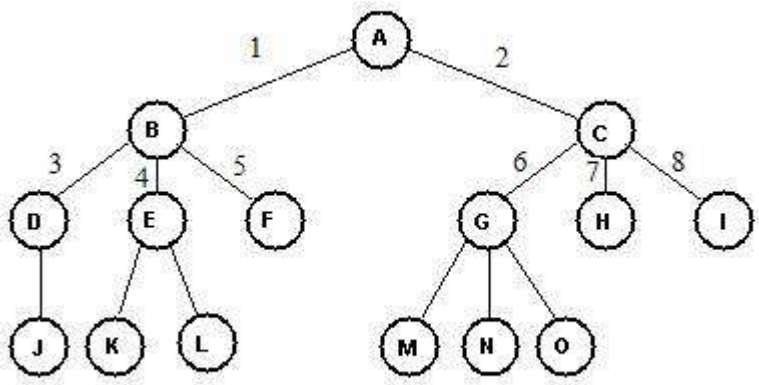
3. Vyber z čela fronty OPEN první uzel.

4. Je-li vybraný uzel uzlem cílovým, ukonči prohledávání jako úspěšné a vrať cestu od kořenového uzlu k uzlu cílovému (vrací se posloupnost stavů, nebo operátorů). Jinak pokračuj.

5. Vybraný uzel expanduj, všechny jeho bezprostřední následníky umísti do fronty OPEN a vrať se na bod 2.

více informací naleznete v [1].

Jak již název napovídá, tato metoda se k výsledku dostává prohledáváním nejbližších stavů, do kterých může expandovat. Postup této metody při prohledávání stromového grafu:

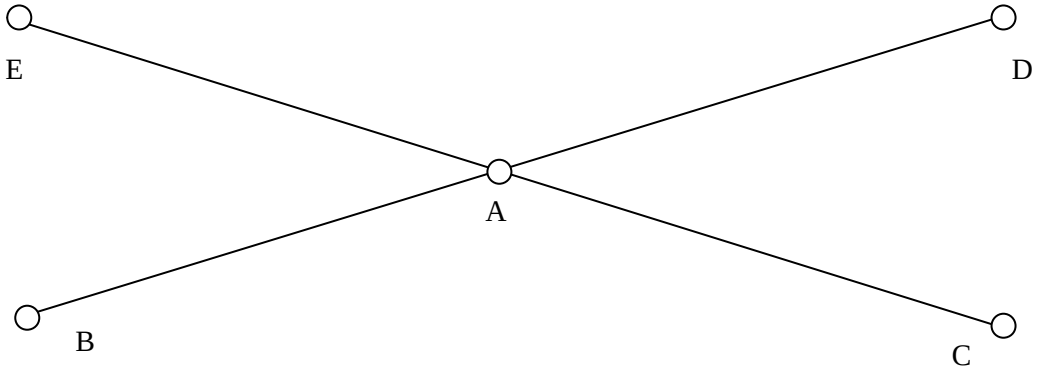


Obr. 3 Metoda BFS na stromovém grafu

Výsledné řešení bylo nalezeno v osmém kroku. Z obrázku je patrné, že pokud by hledaným stavem byl uzel „O“, byl by tento stav nalezen až jako poslední. Algoritmus této metody je možné modifikovat a snížit tím její paměťovou náročnost, avšak těmito modifikacemi se dále nebudu zabývat. Tato metoda i po jejích modifikacích nedosahuje pro tento problém takových kvalit, jako metody dále uvedené.

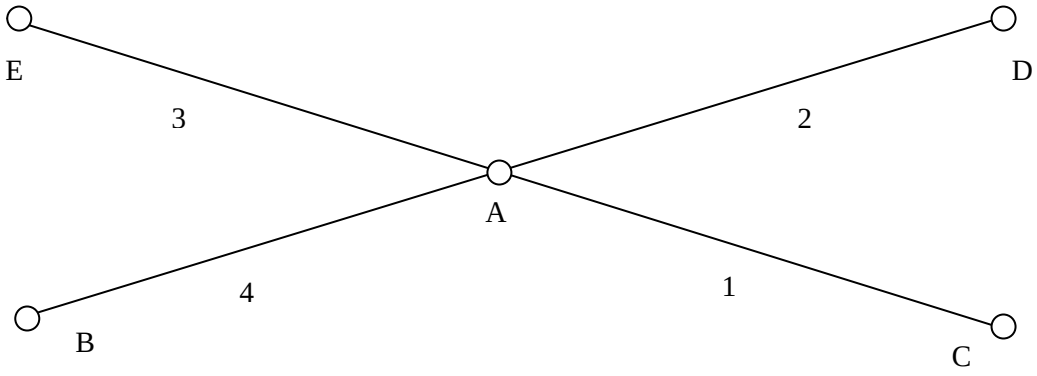
Použitelnost metody:

Tato metoda je pro tuto práci použitelná. Při vhodně zaznamenanými údaji o chodbách a křižovatkách fakulty (hranách a uzlech grafu) najde vždy nejkratší, a tudíž požadovanou trasu. Je však značně neefektivní. Viz následující obrázek.



Obr. 4 Jednoduchý graf

V případě, že bychom se chtěli dostat z uzlu A do uzlu B. V nejhorším případě (předpokládejme, že uzly expanduje náhodně) bude algoritmus postupovat následovně:



Obr. 5 Metoda BFS na jednoduchém grafu

Z obrázku je jasné, že tato metoda sice nejkratší cestu našla, ale to až jako poslední čili značně neefektivně. A vzhledem k tomu, že model budovy je systém částečně známý, je vhodnější využít jiné rodiny algoritmů.

2.1.2 Metoda slepého prohledávání do hloubky (DFS)

Základní algoritmus metody DFS je následující:

1. Sestroj zásobník OPEN (bude obsahovat všechny uzly určené k expanzi) a umísti do něj počáteční uzel.

2. Je-li zásobník OPEN prázdný, pak úloha nemá řešení a ukonči proto prohledávání jako neúspěšné. Jinak pokračuj.

3. Vyber z vrcholu zásobníku OPEN první uzel.

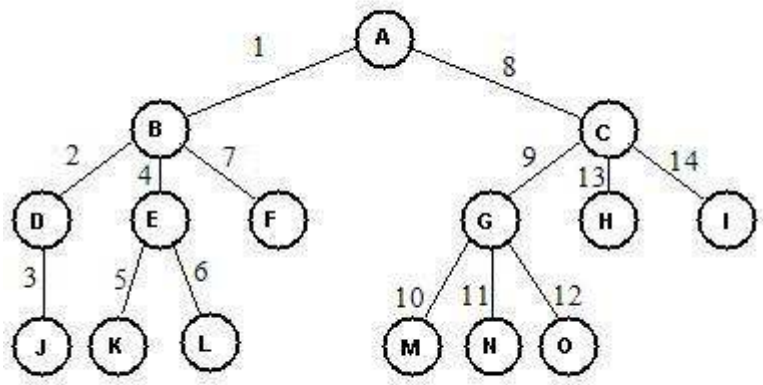
4. Je-li vybraný uzel uzlem cílovým, ukonči prohledávání jako úspěšné a vrať cestu od kořenového uzlu k uzlu cílovému (vrací se posloupnost stavů, nebo operátorů). Jinak pokračuj.

5. Vybraný uzel expanduj, všechny jeho bezprostřední následníky umísti do zásobníku OPEN a vrať se na bod 2.

Algoritmus DFS není, na rozdíl od výše popsaného algoritmu BFS ani úplný, ani optimální. Časová náročnost algoritmu DFS zůstává exponenciální – teoreticky je dána výrazem O(bm), kde b je faktor větvení a m je maximální prohledávaná hloubka stromu.

více informací naleznete v [1].

Mějme stejný problém jako u metody BFS. Ve stromovém grafu s počátečním uzlem „A“ hledejme uzel „I“. Metoda DFS bude postupovat následovně:

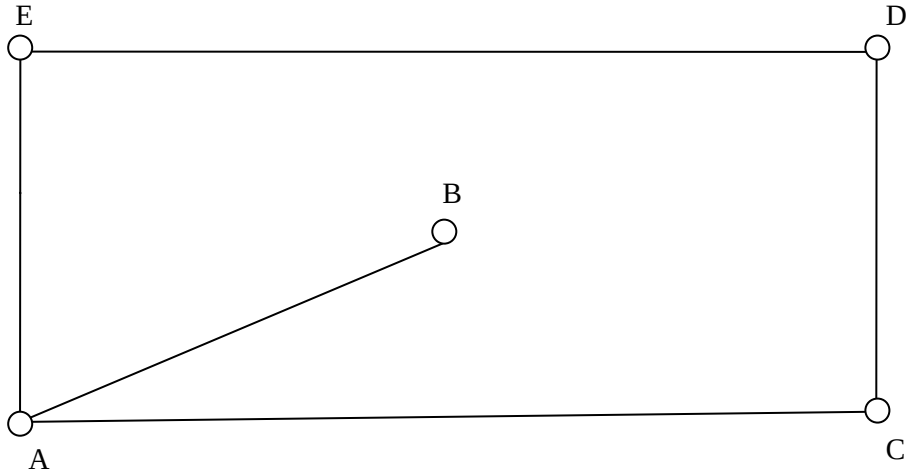


Obr. 6 Metoda DFS na stromovém grafu

Algoritmus metody prohledává slepě do hloubky, dokud nenarazí na konec, nebo dokud nenajde výsledný stav. V tomto případě nalezl výsledný stav až v posledním kroku. Pokud by však hledaným uzlem byl uzel „J“, nalezl by tento již v třetím kroku, zatímco metoda BFS až v devátém.

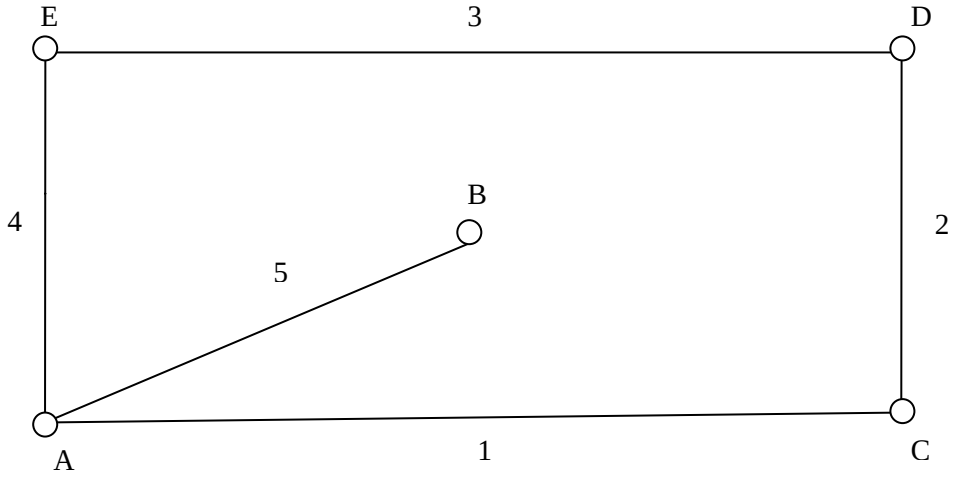
Použitelnost metody:

Opět použijeme jednoduchý graf. Na kterém algoritmus bude hledat nejkratší vzdálenost mezi počátečním uzlem A a konečným uzlem B:



Obr. 7 Jednoduchý graf

Předpokládejme, že se bude řídit algoritmem „do leva“. Jako první uzel pro pokračování si vybere uzel C. Metoda poté bude postupovat následovně:



Obr. 8 Metoda DFS na jednoduchém grafu

Z obrázku je patrné, že tato rodina algoritmů je pro tuto práci jedna z nejnevhodnějších. Zvláště také proto, že graf reprezentující fakultu je cyklický a pro takový je tento algoritmus téměř nepoužitelný. Bez řídících podmínek algoritmus může jednoduše zůstat obíhat jeden cyklus grafu do nekonečna. Další modifikace tohoto algoritmu nijak zásadně nezlepší použitelnost tohoto algoritmu, proto je nebudu uvádět.

Tyto dva výše uvedené algoritmy patří ve své rodině mezi nejzákladnější algoritmy. Jejich použitelnost je výpočtově a také paměťově náročná. Tyto požadavky se ovšem dají regulovat příslušnými úpravami algoritmů. Takovými úpravami vznikly algoritmy jako např.

- **DLS - Depth Limited Search**
- **IDS - Iterative Deepening Search**
- **BS – Bidirectional Search**
- **UCS – Uniform Cost Search**
- **Backtracking**
- **Forward Checking**

Neinformované metody jsou všeobecně vhodné pro prohledávání stavů, o kterých mnoho informací není známo.

V této práci je ovšem pracováno s modelem, o kterém je známo informací poměrně hodně. Proto je vhodné zaměřit se na použití metod informovaných.

2.2 Informované metody

> *Informované metody mají k dispozici a používají nějakou informaci o cílovém stavu a mají prostředky, jak aktuální stavy prohledávání hodnotit. Právě podobné metody převážně používá i člověk – vrátíme-li se k příkladu prohledávání mapy, pak hledáme-li cestu z nějakého (výchozího) místa do jiného (cílového) místa máme obvykle alespoň přibližnou představu, ve kterém směru od výchozího místa cílové místo leží. Je zřejmé, že čím je naše představa o poloze cílového místa přesnější, tím menší oblast mapy musíme prohledávat.*

více informací naleznete v [1].

Mezi informované metody prohledávání grafu patří algoritmy:

- **BestFS – Best First Search**
- **GS – Greedy Search**
- **A* Search**
- **Hill Climbing Search**
- **Simulated Annealing**
- **Heuristic Repair**

V této aplikaci budeme o budovách fakulty uchovávat informace o vzájemné poloze místností i jejich podlaží. V případě otevřených prostor (např. nádvoří) bude tento prostor reprezentován spojnicemi všech jeho sousedících uzlů (vchody/východy).

2.2.1 Metoda BestFS

Podívejme se nyní na algoritmus informované metody **BestFS**

1. Sestroj seznam OPEN (bude obsahovat všechny uzly určené k expanzi) a umísti do něj počáteční uzel včetně jeho ohodnocení.

2. Je-li seznam OPEN prázdný, pak úloha nemá řešení a ukonči proto prohledávání jako neúspěšné. Jinak pokračuj.

3. Vyber ze seznamu OPEN uzel s nejlepším (nejnižším) ohodnocením.

4. Je-li vybraný uzel uzlem cílovým, ukonči prohledávání jako úspěšné a vrať cestu od kořenového uzlu k uzlu cílovému (vrací se posloupnost stavů, nebo operátorů). Jinak pokračuj.

5. Vybraný uzel expanduj, všechny jeho bezprostřední následníky, kteří nejsou jeho předky, umísti do seznamu OPEN, a to včetně jejich ohodnocení. Vrať se na bod 2.

více informací naleznete v [1].

Tento algoritmus se od neinformovaného algoritmu **BFS** (viz. kapitola 3.1.1) téměř neliší. Zásadní rozdíl v těchto algoritmech tvoří způsob ohodnocování hran a uzlů. Zatímco neinformované metody mají pro ohodnocení následujícího uzlu informaci pouze o hodnotě hrany (např. délka hrany), metody informované mají k ohodnocování informací více (např. délka a směr hrany). Funkci pro ohodnocení hrany budeme označovat jako**:**

Optimální funkce

Tato optimální funkce poté bude rozhodovat, které uzly budou prioritně určeny k expanzi a v jakém pořadí budou ukládány do struktur OPEN algoritmů.

2.2.2 Metoda A*

Metoda A je nejznámější a nejpoužívanější metodou pro řešení úloh prohledáváním stavového prostoru. K ohodnocení uzlů používá vztah f(n) = g(n) + h(n), kde heuristická funkce h(n) musí být tzv. spodním odhadem skutečné ceny cesty od ohodnocovaného uzlu k cíli – taková heuristika se pak nazývá přípustnou heuristikou (resp. heuristickou funkcí).*

Metoda A je úplná a pro přípustné heuristické funkce je i optimální. Důkaz optimálnosti je poměrně jednoduchý:*

1. Označme cenu optimální cesty do optimálního cílového uzlu jako fo (sGopt) a cenu jiné (neoptimální) cesty do stejného nebo jiného cílového uzlu jako f(sG). Zřejmě musí platit
fo (sGopt) < f(sG).

2. Nechť x je uzel na optimální cestě k optimálnímu cíli. Pro přípustnou heuristiku musí platit
fo (sGopt) ≥ f(x).

3. Pokud by uzel x nebyl vybrán k expanzi, zatímco cílový uzel sG s ohodnocením f(sG) ano, muselo by platit f(x) ≥ f(sG).

4. Z bodů 2 a 3 vyplývá, že fo (sGopt) ≥ f(x) ≥ f(sG), tedy že fo (sGopt) ≥ f(sG), což je ve sporu se závěrem bodu 1.

5. Uzel x proto musí být vybrán k expanzi, stejně jako každý jiný uzel na optimální cestě k optimálnímu cíli a metoda proto musí nalézt optimální cestu.

více informací naleznete v [1].

Algoritmus A* expanduje pouze uzly x pro které platí f(sx) £ fo, kde fo je cena optimální cesty. Časovou a prostorovou náročnost proto výrazně ovlivňuje použitá heuristika – pokud se blíží 0 (to je jistě spodní odhad skutečné ceny), pak složitost se blíží exponenciální složitosti, pokud je použitá heuristika dobrým spodním odhadem skutečné ceny, pak jsou expandovány pouze uzly kolem optimální cesty (je-li přesným odhadem, pak jsou expandovány pouze uzly na optimální cestě).

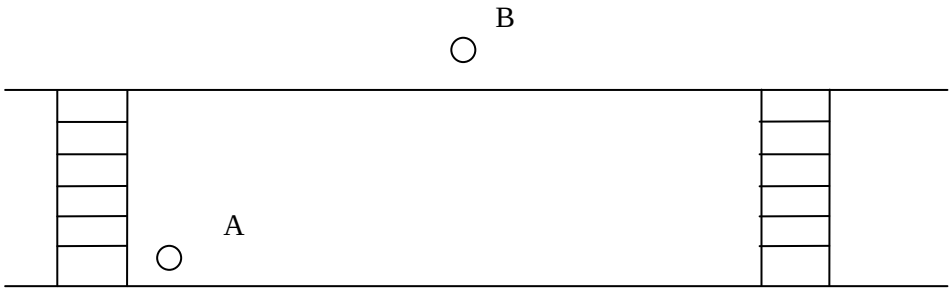
Tato metoda dále nijak nevylepšuje vyhledávací algoritmus. Ten, co se informovaných metod týče, zůstane v základu stejný. Další vývoj se zaměřuje na ohodnocování hran grafu. Z této informované metody jsem vycházel při vytváření výsledného algoritmu.

2.3 Výsledný FIT algoritmus

Mezi důležité informace o grafu reprezentujícím budovu je informace o podlaží hledaných uzlů. Ovšem přemisťování se v rámci podlaží s sebou přináší určitý problém. Díky limitovaným

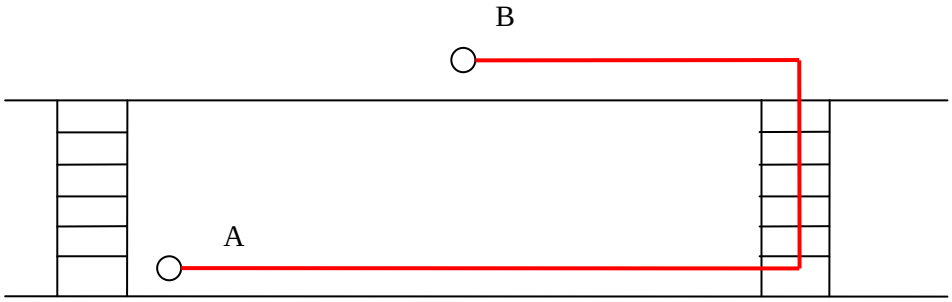
počtům přechodů mezi patry se může stát, že algoritmus nenajde nejkratší cestu. Tento problém demonstrujeme na následujícím obrázku.

Mějme dva uzly grafu, o kterých známe jejich polohu. (hledáme cestu z A do B)



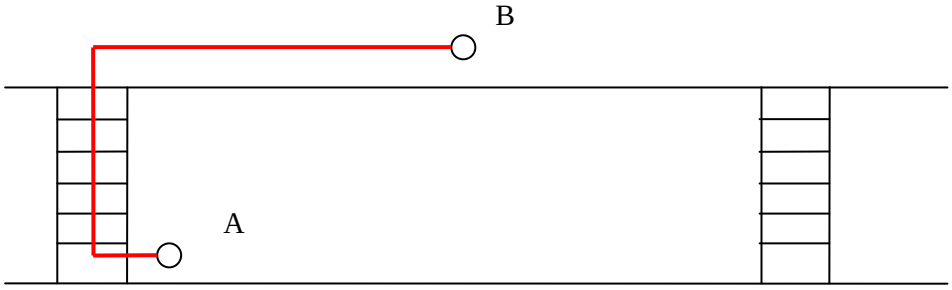
Obr. 9 Problém přechodu mezi patry I

Nejjednodušší algoritmus, který na základě polohy vyrazí směrem k cílovému uzlu, nalezne následující cestu.



Obr. 10 Problém přechodu mezi patry II

Tato cesta samozřejmě nejkratší není. Nejkratší cesta vypadá následovně:



Obr. 11 Problém přechodu mezi patry III

Tato skutečnost je důležitá při vytváření konečného algoritmu.

Jak už bylo řečeno, vycházíme z grafu, kde o jeho uzlech uchováváme informace o jejich vzájemných polohách a o podlažích, na kterých jsou tyto umístěny. Stejně tak o hranách grafu aplikace uchovává informace jak o jejich délkách tak i směrech.

S takto podrobnými informacemi o prohledávaném prostoru se nabízí velice jednoduchý algoritmus pro vyhledání nejkratší cesty. Předpokládejme, že každý uzel grafu má minimálně jednu hranu:

1. Umísti do seznamu OPEN počáteční uzel včetně jeho ohodnocení.

2. Je-li seznam OPEN prázdný, pak úloha nemá řešení a ukonči proto prohledávání jako neúspěšné. Jinak pokračuj.

3. Vyber ze seznamu OPEN uzel s nejlepším (nejnižším) ohodnocením.

4. Je-li vybraný uzel uzlem cílovým, ukonči prohledávání jako úspěšné a vrať cestu od kořenového uzlu k uzlu cílovému. Jinak pokračuj.

5. Vybraný uzel expanduj, všechny jeho následníky, co se přibližují k cílovému uzlu, umísti do seznamu OPEN, a to včetně jejich ohodnocení. Vrať se na bod 2.

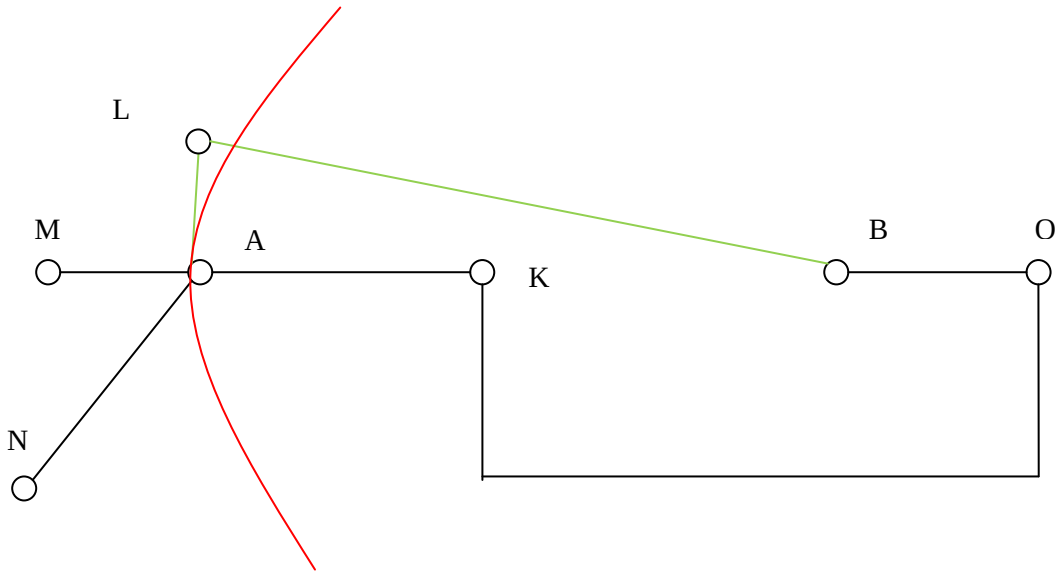
Tento algoritmu by však fungoval pouze v rámci jednoho podlaží jedné budovy. A to také ne vždy úplně správně. Výše popsaný problém při přesunu mezi podlažími je aktuální i např. při budově tvaru čtverce. Princip problému zůstává ve skutečnosti, že nejkratší trasa obsahuje úseky, které jsou ohodnoceny záporně (jinými slovy, musíme se kousek vrátit, abychom vyrazili rychleji). Pojmenujme tento problém jako „**Problém potřebného couvání**“.

2.3.1 Problém potřebného couvání

Prvním řešením, které jsem se pokusil aplikovat v praxi, bylo zavedení určité povolené odchylky od trasy. Tzv.
faktor couvání

značme Fc. Tento faktor jednoduše povolil algoritmu uznat určité cesty, které by jinak neuznal.

Příklad. Předpokládejme, že algoritmus se snaží najít nejkratší trasu z uzlu A do uzlu B. Expanduje pouze uzly, které se přibližují svým umístěním cílovému uzlu. A že graf je nakreslen v klasickém souřadném systému, tedy ohodnocení hran odpovídá jejich nakresleným délkám. Z obrázku je tedy patrné, že nejkratší cestu k cílovému uzlu tvoří posloupnost uzlů A,L,B.



Obr. 12 Graf bez faktoru couvání

Nicméně díky podmínce, že expandovat se budou jen uzly přibližující se k cíly, vyhodnotil by jako nejkratší cestu posloupnost uzlů A –> K –> O –> B.

Uzly L, M a N by vůbec nevzal v úvahu. To ovšem není chtěný postup. Proto do výpočtu ohodnocení uzlu W:

ohodnocovaný uzel = O
počáteční uzel = A
koncový uzel = B
vzdálenost uzlů A a B = |AB|
Přidáme faktor couvání:

$$W = |AB| - |OB|$$

$$W = |AB| - |OB| + Fc$$

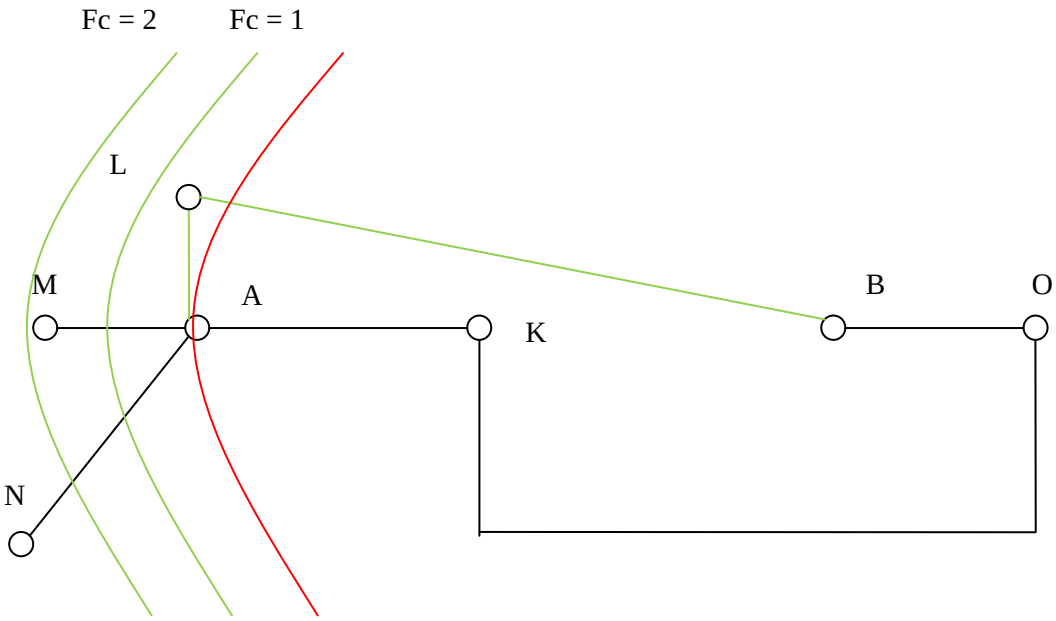
podmínka expanze je poté definována vztahem:

$$\underline{W > 0}$$

Faktorem couvání rozšíříme pole akceptovatelných uzlů pro expanzi.

Uzel tedy může expandovat v případě, že splňuje *podmínku expanze*, tedy že se svou polohou přibližuje cílovému uzlu, nebo nepřesahuje svou polohou pole akceptovatelných uzlů pro expanzi rozšířeným *faktorem couvání.*

Příklad:

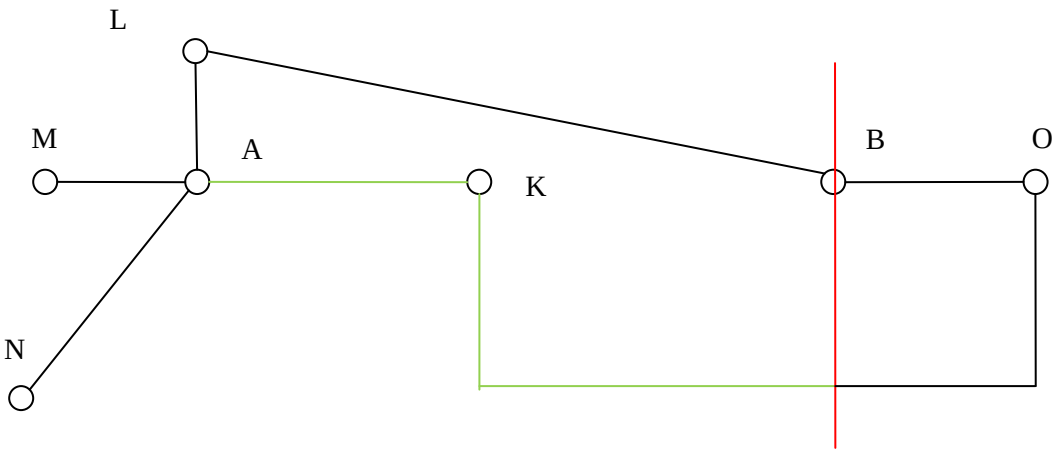


Obr. 13 Graf s aplikovaným faktorem couvání

Tedy pokud by faktor couvání byl dostatečně velký pro akceptaci uzlu L, algoritmus by našel nejkratší možnou trasu. Nicméně, jak obrázek napovídá, ani tato metoda není stoprocentním řešením tohoto problému. Pokud by vedla nejkratší cesta přes uzel M, musel by se dostatečně zvýšit i faktor couvání.

Nikdy totiž nevíme, jak moc musíme couvat, abychom našli optimální řešení. A navíc s rostoucí hodnotou tohoto faktoru také rapidně klesá efektivita vyhledávacího algoritmu.

Druhé řešení tohoto problému vychází ze skutečnosti, že takováto situace v grafech reprezentujících budovy nastává málokdy. Tedy že algoritmus tento problém bude ignorovat, dokud se v konstrukci trasy v jednom kroku nedopustí prodloužení trasy.Jedná se konkrétně o tento okamžik:



Obr. 14 Okamžik prodloužení cesty v grafu

Jedná se o drobnou úpravu algoritmu, kdy tento kontroluje s každým krokem, zda tato skutečnost nenastala. Pokud skutečnost nastane, trasa se přepočítá algoritmem, který bude expandovat pro všechny uzly bez výjimky.

Tento postup byl po sérii testování upraven o faktor couvání. Jelikož bez tohoto omezení se algoritmus při porušení podmínky prodloužení cesty choval téměř jako slepý algoritmus DFS. Faktor couvání je vypočítán při zadání startovního a cílového uzlu jako 40% vzdálenosti mezi těmito dvěma uzly.

Trasy, které vyžadují více jak 40% couvání jejich celkové délky, by se v moderní architektuře neměly objevit. Nicméně v případě, že by se přece jen objevily, například vlivem rekonstrukce budov a tudíž neprostupností některých klíčových spojů či nadměrnou kreativitou architektů, provede se přepočet trasy s neomezeným faktorem couvání, tzn. Algoritmem téměř identickým slepé metodě DFS.

2.3.2 Algoritmus jedné budovy

Tímto dostáváme konečný *algoritmus jedné budovy* pro vyhledávání trasy v rámci jedné budovy. Je možno zapsat:

1. Sestroj a vyčisti seznam OPEN a umísti do něj počáteční uzel včetně jeho ohodnocení.

2. Je-li seznam OPEN prázdný a je nastaven příznak „DFS“ pak úloha nemá řešení a ukonči proto prohledávání jako neúspěšné. Je-li seznam OPEN prázdný a není nastaven příznak DFS pak nastav příznak „DFS“ a pokračuj bodem 1. Jinak pokračuj.

3. Vyber ze seznamu OPEN uzel s nejlepším ohodnocením.

4. Je-li vybraný uzel uzlem cílovým, ukonči prohledávání jako úspěšné. Pokud toto není první úspěšné řešení, porovnej výsledek s nejkratší trasou a pokud je toto hledání trasou nejkratší, trasu si zapamatuj. Pokud se jedná o první úspěšné řešení, trasu si zapamatuj. V případě úspěšného řešení pokračuj bodem 2, jinak pokračuj.

5. Vybraný uzel expanduj. Dodržuj pravidla pro expanzi uzlu. Z uzlů, které se v seznamu OPEN vyskytují vícekrát, ponech pouze uzel s nejlepším ohodnocením, ostatní se seznamu OPEN vyškrtni, a vrať se na bod 2. Pokud při expanzi došlo k prodloužení trasy, nastav příznak „couvání“ a pokračuj bodem 1.

Pravidla pro expanzi uzlu:

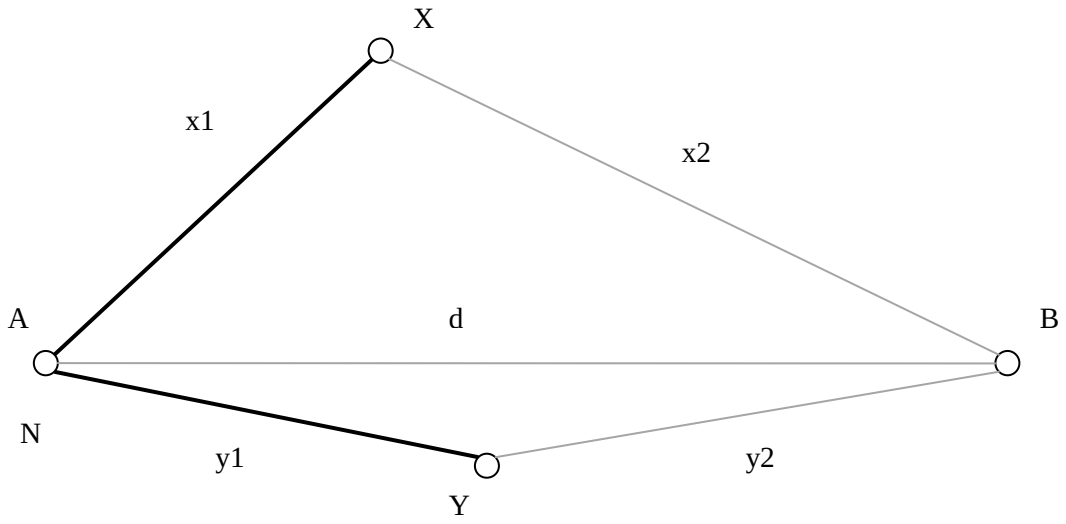
1. Fc (faktor couvání) nastav na 0.
2. Pokud je nastaven příznak „couvání“, nastav Fc na 40% vzdálenosti počátečního a cílového uzlu.

3. Pokud je nastaven příznak „DFS“, nastav Fc na 400% vzdálenosti počátečního a cílového uzlu.
4. Pokud alespoň jeden následník tohoto uzlu splňuje *podmínku expanze* (jako startovní uzel se dosazuje aktuálně expandovaný uzel, cílový zůstává po celý průběh algoritmu konstantní), je možné uzel expandovat.

Podmínka expanze pro uzel je popsána výše (kapitola 2.3.1). Zbývá definovat výpočet ohodnocení hrany grafu tedy *optimální funkce* grafu.

Optimální funkce:

Je vypočítána jako poměr délky hrany vůči délce uražené trasy. Vysvětlení viz obrázek:



Obr. 15 Ohodnocení hran grafu

d…vzdálenost mezi počátečním a koncovým uzlem grafu
x1…délka první zkoumané hrany
x2…vzdálenost mezi koncovým uzlem grafu a koncovým uzlem první zkoumané hrany
y1…délka druhé zkoumané hrany
y2…vzdálenost mezi koncovým uzlem grafu a koncovým uzlem druhé zkoumané hrany

optimální funkce C např. pro hranu X se poté vypočítá ze vztahu:

$$C = d / (x1 + x2)$$

$$C \in \langle 0, 1 \rangle$$

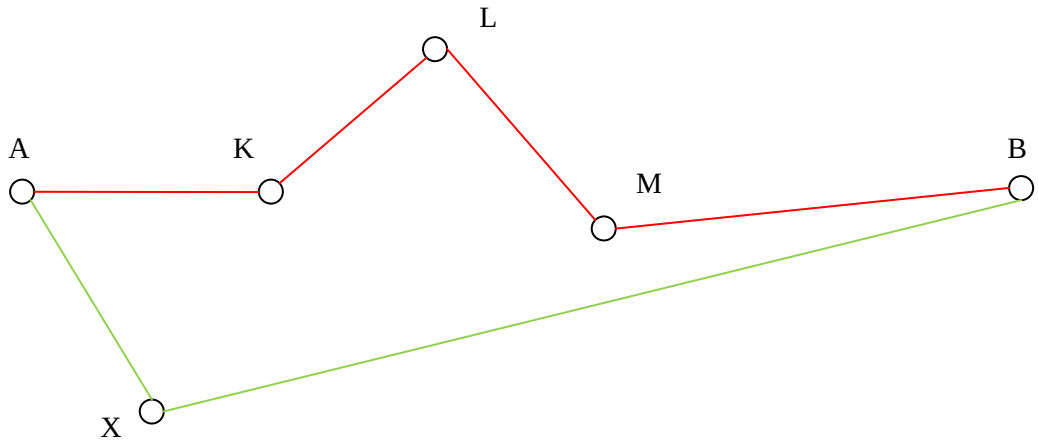
Tedy *čím větší ohodnocení tím lepší*. Ideální ohodnocení je rovno jedné. V tomto případě z obrázku platí:

$$Cy > Cx$$

Jako prioritní následník uzlu A se tedy vybere uzel Y. Pokud uzel X splňuje podmínku expanze, uloží se do fronty OPEN jako druhý.

Jednoduše řečeno, algoritmus expanduje pro uzly, které se přibližují k cíli. Z těchto cest (ve většině případů nejsou více než dvě) poté vybere tu nejkratší. Pokud při hledání narazí na překážky (rekonstrukce, prodloužení trasy), řídí se výše popsanými pravidly.

Proč algoritmus bere v úvahu vícero možných cest, a ne jen jednu optimální, když vychází z algoritmu A*, který expanduje jen pro uzly na nejkratší cestě? Tato skutečnost je zapříčiněna podobným principem, jako nutnost zavedení faktoru couvání. A to sice, že cesta nejkratší trasy může obsahovat uzly, které na určitých křižovatkách nejsou nejoptimálnější. Nejlépe bude si tento stav opět demonstrovat na obrázku. Opět jako startovní a cílový uzel budou použity uzly A a B.



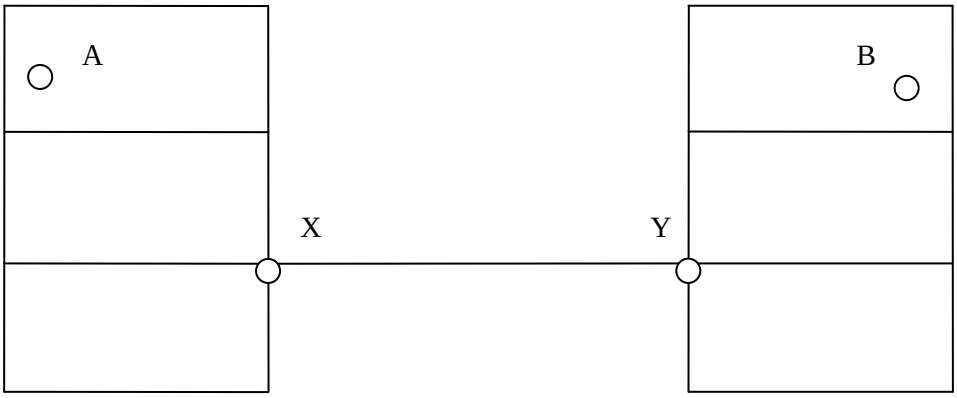
Obr. 16 Více cest v grafu

Z obrázku to dost dobře vidět není, ale cesta přes uzel X (posloupnost A, X, B) je kratší. Nicméně při expanzi uzlu A by sice podmínku expanze splnily jak uzly X tak K, ale uzel K by byl díky jeho vyšší optimální funkci vybrán prioritně. Jelikož by všechny následující uzly uzlu K splnili podmínku expanze, algoritmus by skončil úspěšně a vyhledávání neúspěšně. Proto algoritmus prověří všechny cesty složené z uzlů, které splňují podmínku expanze (přibližují se cílovému uzlu). V praxi situace z obrázku opět nastává málokdy, avšak pro bezchybné fungování algoritmu je toto opatření nutné.

Toto je konečný algoritmus pro nejrychlejší vyhledávání nejkratší cesty v rámci jedné budovy. Což bohužel není náš případ, jelikož chceme algoritmus pro vyhledávání v rámci celého areálu budov. Takové vyhledávání s sebou přináší další problém. A to problém přechodu mezi budovami.

2.3.3 Problém přechodu mezi budovami

Tento problém není těžké si představit. Mějme dvě budovy o třech podlažích, spojené jedním můstkem umístěným v prvním podlaží. Hledat se bude trasa z místnosti A ve třetím patře první budovy do místnosti B ve třetím patře druhé budovy.



Obr. 17 Problém přechodu mezi budovami

V této situaci nastává problém. Oba dva uzly grafu se nachází ve stejném podlaží, avšak v rámci těchto podlaží mezi nimi neexistuje spojnice. Opět se nabízí dvě řešení tohoto problému.

První možností řešení je rozšířit pravidla a výpočty pro expanzi a ohodnocování hran grafu z rovinných výpočtů v rámci jednoho podlaží do prostorových výpočtů v rámci celého grafu (v tomto případě areálu). Tímto krokem se ovšem celá náročnost aplikace na výpočty a paměť rapidně zvýší. V neposlední řadě by tento krok také zvýšil procento případů uchýlení se k prohledávání s neomezeným faktorem couvání z mizivé hodnoty na nepříjemně nepřehlédnutelnou hodnotu (v závislosti na možnostech propojení budov).

Druhá možnost řešení vychází z prosté a všeobecně rozšířené poučky: „Těžký úkol si rozděl na více menších úkolů“. Neboli vyhledávej vícekrát.

V tomto konkrétním případě z obrázku o dvou budovách a jednom můstku by se tedy nevyhledala přímo trasa z uzlu A do uzlu B, ale proběhly by dva algoritmy pro výpočet dvou tras:

- Trasa z uzlu A do uzlu X.
- Trasa z uzlu Y do uzlu B.

Tímto krokem se zachová rychlost a jednoduchost aplikace. K docílení tohoto stavu je však nutné zavést do grafu nový speciální druh uzlu, tzv. *Bridge uzel*.

2.3.3.1 Bridge uzel

Je takový uzel, který má kromě standardních vlastností uzlu grafu také nějaké informace navíc. V aplikaci jsou poté tyto informace o dvojicích takovýchto uzlů uchovávány v *tabulce bridge uzlů*. Každý bridge uzel si uchovává navíc seznam uzlů, ke kterým v rámci své budovy vede.

Tabulka bridge uzlů obsahuje seznam struktur, které obsahují dvojice bridge uzlů, a ke každému bridge uzlu uchovávají názvy uzlů, ke kterým tento uzel vede. Tato struktura se dá nejlépe reprezentovat následující tabulkou (pro příklad z obrázku):

X	Y
A	B
…	…

Tabulka 1. Dvojice bridge uzlů

Jelikož se jedná pouze o textové informace (unikátní klíče uzlů grafu) a takovýchto „mostů“ se v areálech nevyskytuje mnoho, rozhodl jsem se v mé aplikaci využít tohoto řešení.
Použitím bridge uzlů vznikne následující nadstavba nad *algoritmem jedné budovy*:

1. Vyhledej v tabulce bridge uzlů počáteční uzel trasy.

2. Pokud se cílový uzel nachází ve stejné budově jako uzel počáteční, proveď výpočet trasy pro tyto dva uzly. Jinak pokračuj.

3. Vyhledej v tabulce bridge uzlů seznam bridge uzlů, kterými je nutno projít.

4. Proveď výpočet tras pro sekvenci těchto bridge uzlů a sestav kompletní trasu.

Tyto algoritmy využívá výsledná aplikace pro navigaci v areálu fakulty informačních technologií.

3 Aplikace

Aplikace byla vyvinuta v prostředí MS Visual studio 2005. Je napsána v programovacím jazyce C# a jedná se o aplikaci typu winform. Jádro aplikace tvoří hlavní formulář, ve kterém se nachází téměř všechny ovládací prvky.

Aplikace využívá pouze základních knihoven a prostředků. Je proto snadno přeložitelná a spustitelná v prostředí Centra výpočetní techniky.

3.1 Seznam souborů:

Aplikace je vytvořena v prostředí MS Visual studio 2005 jako samostatný projekt. Má tudíž implicitně vytvořeno několik souborů, nutných pro kompilaci, tyto nebudu vypisovat. Soubory, které byly přidány mimo automatické vytvoření projektu v MS Visual studiu:

- MapNodeClass.cs – tento soubor definuje třídu pro uzly grafu se všemi vlastnostmi a funkcemi, které se nad těmito uzly mohou provádět.

- ArealClass.cs – definuje třídu pro areál budov. Objekt této třídy v aplikaci existuje pouze jeden. Tento objekt obsahuje všechny důležité vlastnosti pro aplikaci, seznamy všech uzlů grafu, tabulku bridge uzlů, atd. Tato třída také definuje algoritmus pro zjištění a vykreslení nejkratší trasy v grafu.

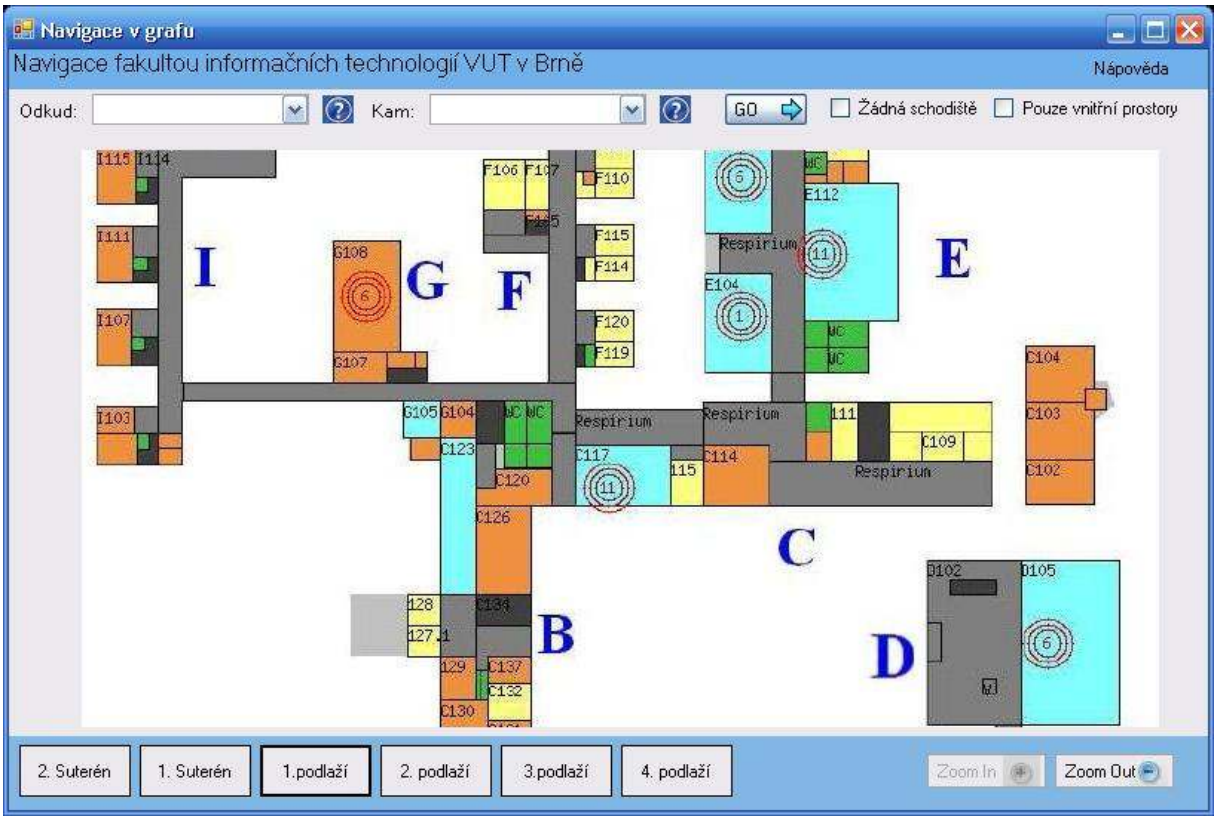
- MainFrm.cs – definuje hlavní formulář aplikace, chování ovládacích prvků. Tento formulář řídí chod celé aplikace.

- HelpFrm.cs – tento soubor obsahuje formulář s nápovědou a s informacemi o aplikaci a autorovi.

- InfoNodeFrm.cs – obsahuje formulář, který zobrazí informace (pokud nějaké dostupné jsou) o vybraném uzlu a umožní jejich případnou editaci. (Neprůchodnost vchodů, nefunkční výtah,…)

3.2 API

(APlication Interface) Uživatelské rozhranní aplikace je tvořeno hlavním formulářem. Hlavní formulář, tedy ten, který se zobrazí po spuštění aplikace vypadá následovně:



Obr. 18 Hlavní formulář aplikace.

Jak je patrno, inspiraci pro vzhled aplikace jsem čerpal právě na stránkách fakulty informačních technologií. Také odtud jsem čerpal obrázky pro plány a popis místností, chodeb i samotných budov. Stejně jako tyto stránky jsem se snažil aplikaci udržet laděnou do modré barvy.

Nyní tedy popis ovládání aplikace. Nejdominantnější část tohoto formuláře tvoří obrázek plánu fakulty. Tento je jako jediný ovládán dynamicky, a to pohybem myši. Tento ovládací prvek je převzatý z všeobecně rozšířeného mapového portálu (www.mapy.cz). Ovládání obrazu se nikterak neliší, pomocí stisknutého levého tlačítka myši a pohybu myši dochází k pohybu celého obrazu podlaží fakulty. Tento ovládací prvek jsem zvolil vzhledem k jeho všeobecné rozšířenosti mezi uživateli a tudíž k přiblížení uživatelsky příjemnému chování aplikace.

Ostatní ovládací prvky jsou v této aplikaci statické. Tzn., Jsou tvořeny statickými komponentami a jejich ovládání je od defaultně nastaveného nezměněno, takže je velice jednoduché. Pokud budeme postupovat shora, pak jsou tyto prvky tvořeny:

Nad plánem fakulty:

- Nápověda – tlačítko nápovědy v pravém horním rohu aplikace. Zobrazí formulář s informacemi o aplikaci a jejím autorovi.

- Odkud – textové pole v levém horním rohu aplikace. Slouží pro výběr či zadání startovního uzlu ze seznamu povolených uzlů.

- Kam – textové pole pro výběr cílového uzlu. Viz. Odkud.

- Informace – Napravo od těchto dvou textových polí jsou umístěná tlačítka o informacích o zadaných uzlech. Tyto tlačítka zobrazí formulář pro náhled a editaci dostupných informací o zadaném uzlu. Tyto tlačítka obsahují bílý otazník v modrém poli.

- GO – toto tlačítko (pokud jsou zadány počáteční a konečný bod trasy) spustí algoritmus vyhledání nejkratší trasy. A tento výsledek vykreslí do plánu fakulty.

- Schodiště, venkovní prostory – tyto dvě komponenty slouží jako možnost přidat doplňující podmínky do vyhledávacího algoritmu. Mají vliv na vybírání uzlů k expanzi.

Pod plánem fakulty:

- Podlaží – tyto tlačítka, situované v levém spodním rohu hlavního formuláře aplikace, umožňují přepnout náhled na různá podlaží areálu fakulty.

- Zoom in, Zoom out – klasický ovládací prvek navigačních aplikací, slouží k přiblížení-oddálení obrazu plánu fakulty.

Vzhledem k jednoduchosti ovládání aplikace by názorná nápověda spíše znepřehlednila celkový výsledek. Proto formulář s nápovědou obsahuje pouze informace o aplikaci a autorovi. Popis ovládání však bude obsahem stručné dokumentace, která bude na přiloženém DVD.

Obr. 19 O aplikaci

Editačně informační formulář s obsahem informací o jednotlivých uzlech grafu (schodištích, výtazích, vchodech…), spouštěný informačními tlačítky vedle výběru trasy, vypadá následovně:

Obr. 20 Editačně informační formulář

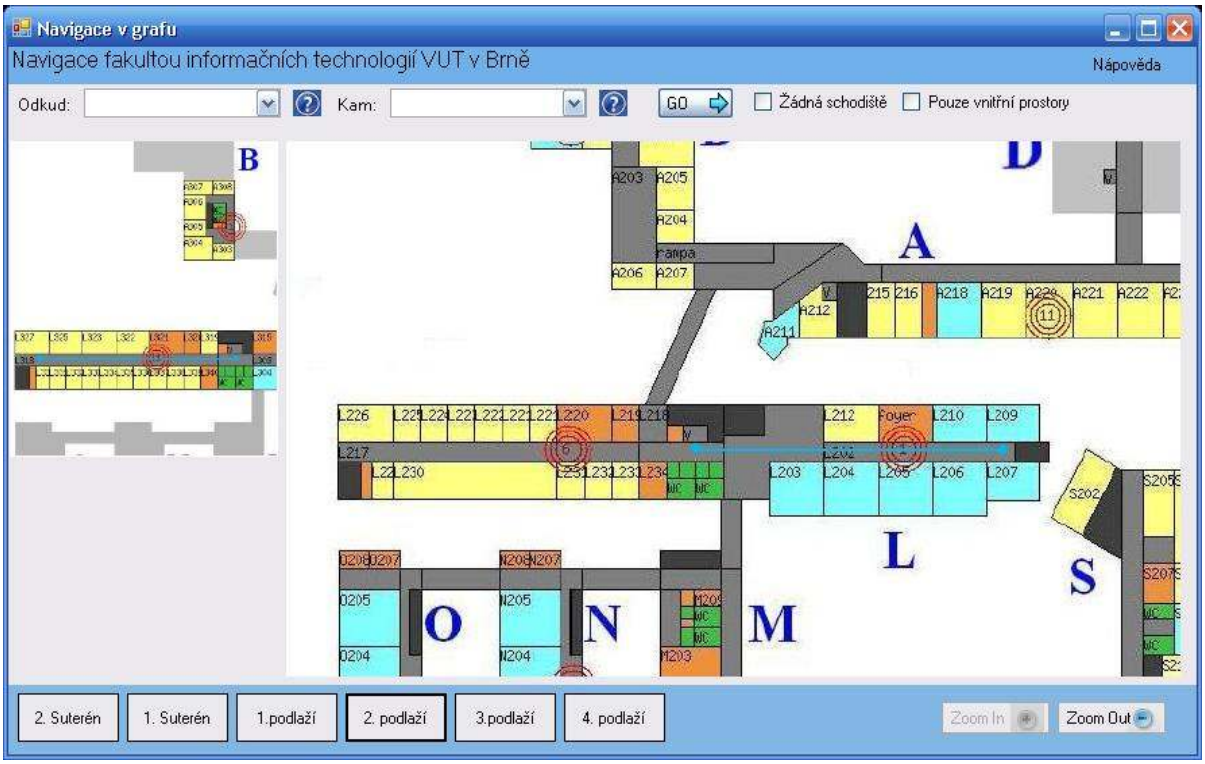
3.3 Budoucí rozšíření

Rozšíření aplikací vnikají z většiny případů na základě zpětných impulzů od konečných uživatelů těchto aplikací. Je to přirozený jev, kdy uživatel vlastně sám, při zadávání požadavku aplikace, neví, co v konečném výsledku chce. Také pokud se aplikace zaměřuje na práci s jevem, který často a zásadně mění svůj charakter (mediální přehrávače vs. nové formáty souborů, účetní systémy vs. reformy daňových zákonů,…), jsou tyto úpravy a rozšíření aplikace nevyhnutelné.

Aplikace založená na teoretické práci, vyvíjená nezkušeným programátorem ve školním prostředí je k těmto úpravám a rozšířením téměř odsouzena. Stejně tak navrhovat rozšíření aplikace na základě testování je poněkud zbytečně předběžné. Nicméně nějaká rozšíření, dle mého názoru praktická, mě přeci jen přišly na mysl. Uvedu dvě dle mého názoru nejpřínosnější.

Zobrazení:

Jedno z vhodných rozšíření aplikace pro budoucnost by nejspíše byla možnost zobrazení více než jednoho podlaží fakulty. Pokud trasa zasahuje do více podlaží, může být přepínání náhledů na tyto různá podlaží poněkud nepřehledné. Možností vytvoření např. menších „náhledů“. Zůstává ovšem otázkou, jaké zvolit ovládání pro tyto náhledy a zda by naopak přehlednosti neuškodily. Viz obrázek znázorňující cestu z třetího podlaží (malý náhled, do druhého podlaží):



Obr. 21 Náhled na více podlaží

Jedním z řešení vícero náhledů je použít systém záložek. Avšak tento systém už v aplikaci implementován je. Dalším možným řešením by mohlo být rozšíření editačních prvků těchto náhledů (zoom, velikost, umístění v aplikaci). Tedy že by si dominantní prostor aplikace utvářel uživatel dle svých potřeb a libostí.

Editor:

Druhým rozšířením, které by znamenalo velký průlom v aplikaci, by určitě byl grafický editor pro tvorbu grafů přímo nad plány budov a areálů.

Tento krok by byl pro aplikaci nejvíce přínosný, avšak také pro implementaci nejvíce náročný. Tento krok by vyžadoval aktualizaci některých ovládacích prvků. Tlačítka pro přepínání jednotlivých podlaží by mohly snadno přesáhnout prostor aplikace, doplňující informace pro vyhledávání nejkratších cest by se museli rozšířit o potřebná data (motorová a jiná doprava v areálech, …) a úměrně množství těchto dat i vrchní ovládací panel aplikace a mnoho dalších hypotetických úprav.

Záměrem této úpravy by bylo upravit aplikaci do takového stádia, kdy by mohla být distribuovatelná bez žádných inicializačních dat, a uživatel by si ji mohl bez zásahu programátora modifikovat k potřebám svým.

Nicméně pomineme-li nutnost úprav ovládacích prvků, stále zůstává velký problém u prvků funkčních.

Bridge uzly, klíčový prvek vyhledávacího algoritmu. Pokud by jejich inicializace byla ponechána v rukou uživatelů, zajisté by tato aplikace byla rychle zavrhnuta z důvodu jejich špatné inicializace a tudíž nefunkčnosti aplikace. Nabízí se alternativní řešení, a to sice, že by si aplikace tyto

uzly vyhledala a inicializovala při tvorbě grafu. Nicméně algoritmus pro vyhledávání a inicializaci těchto uzlů by svou složitostí přesáhl algoritmus vyhledávací, a to masivně.

Z tohoto pohledu, na aplikaci obsahující tento editor, by pravděpodobně změna vyhledávacího algoritmu pro zjednodušení aplikace jako celku, za cenu snížení vyhledávací efektivity, byla na místě.

Pokud se aplikace všeobecně dostane z teoretického využití v rámci fakulty, do praktického využití pro širokou škálu uživatelů, netroufám si odhadnout, zda by mé dosavadní zkušenosti a z nich vyvinutá aplikace obstála.

4 Závěr

V této práci jsem sestavil a aplikoval rychlý a jednoduchý algoritmus pro navigaci v areálu fakulty informačních technologií. Zároveň se mi podařilo při urychlování a konkretizaci tohoto algoritmu udržet dostatečně velkou míru abstrakce k tomu, aby se tento algoritmus dal aplikovat na jakýkoliv areál budov, nebo i budovu samostatnou. Toto je pouze teoretický předpoklad, podložený jen testovacími daty.

Nicméně i když je tento algoritmus použitelný pro jakoukoliv budovu či areál budov, aplikace samotná je zatím úzce specifikovaná pro naši fakultu. Tato skutečnost je dána synchronizací informací o poloze budov vůči sobě a synchronizací vykreslovaných grafů vůči plánům těchto budov. Export a import těchto informací do a z aplikace není problém, avšak kvantita těchto informací spolu s nutností textové reprezentace informací grafického formátu činí tento proces příliš náročný.

Proto by jako první krok v případném budoucím vývoji této aplikace byl nesporně grafický editor umožňující zadávat informace o infrastruktuře budov přímo do jejích plánů. Implementací takového editoru by se použitelnost této aplikace přesunula z kategorie „zákaznický software“ do kategorie „generický software“. Což by byl velký přínos pro tuto práci.

Pokud by se tento krok provedl, následující vývoj aplikace by se pravděpodobně dále řídil z informací poskytnutých zpětnou vazbou uživatelů aplikace (např. studentů). Na základě těchto informací by se aplikace dočkala mnoha aktualizací a úprav. V neposlední řadě si troufám odhadnout, že by nedotčený nezůstal ani samotný vyhledávací algoritmus.

Další rozšíření si netroufám blíže specifikovat bez zpětné vazby uživatelů. Pravděpodobně by se tyto úpravy týkaly vzhledu aplikace, funkčnosti a četnosti ovládacích prvků,…

Grafický vzhled, který nyní vychází z grafického ladění webových stránek fakulty informačních technologií, by pravděpodobně bylo vhodné implementovat volitelně. Aby si uživatelé mohli vzhled definovat podle svých barevných vzorů.

V neposlední řadě by našel využití modul, pro komunikaci přes internet. Který by eliminoval nutnost mít aplikaci v počítači uživatele. Kde by aplikace pracovala na serveru a klient-uživatel by

dosažené výsledky a informace čerpal prostřednictvím internetového prohlížeče. Takováto aplikace by byla vhodným prezentačním nástrojem pro mnoho společností.

Co se zadání bakalářské práce týče, všechny body zadání byly splněny. Algoritmus navržené navigace dosahuje dobrých a stabilních výsledků, aplikace nabízí využitelný prostor pro vývoj, jsem spokojen.

Děkuji za Váš čas při čtení této práce.

Literatura

[1] Zbořil, F.V., Zbořil, F.: Základy umělé inteligence, Učební texty, 2006.

[2] McGraw-Hill: Discrete Mathematics and Its Applications, New York, 2007, ISBN 0-07-288008-2

[3] Gross, Jonathan L.: Handbook of Graph Theory, CRC Press, 2004, ISBN 1-58488-090-2

Seznam příloh